Name: Mohit Kerkar Div: D15C Roll No: 23

Adv DevOps Lab Exp 03

Aim: To understand the Kubernetes Cluster Architecture, install and Spin Up aKubernetes Cluster on Linux Machines/Cloud

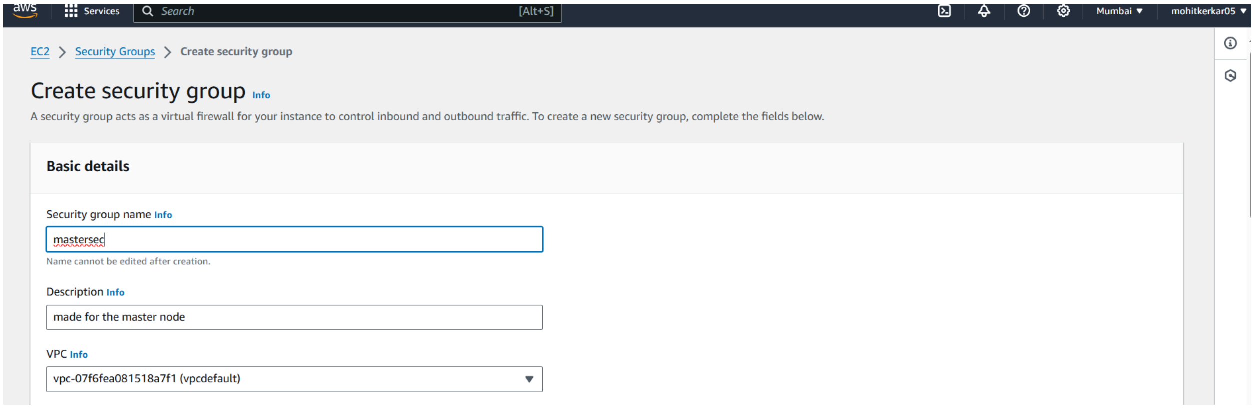
(I have performed this experiment on my personal AWS account)

Step 1: Create Key-pair, Security groups and required default VPCs

I started off by creating two separate security groups i.e one for the master instance (kubeadm to be initialized within it) and the other for the 2 worker instances. For creating any EC2 security groups, we require a VPC (Virtual private cloud) and a subnet along with it .. so that we can work with or allow inbound and outbound traffic and communication. Make sure you have a default VPC and a subnet already created which can be used for this experiment.

Also, make sure that you have a key pair installed of the type RSA with .pem extension on your local machine. Save it at a place which is accessible and where you can work from your terminal.

Here, i created my first security group



EC2 > Security Groups > Create security group

Create security group Info

A security group acts as a virtual firewall for your instance to control inbound and outbound traffic. To create a new security group, complete the fields below.

Basic details

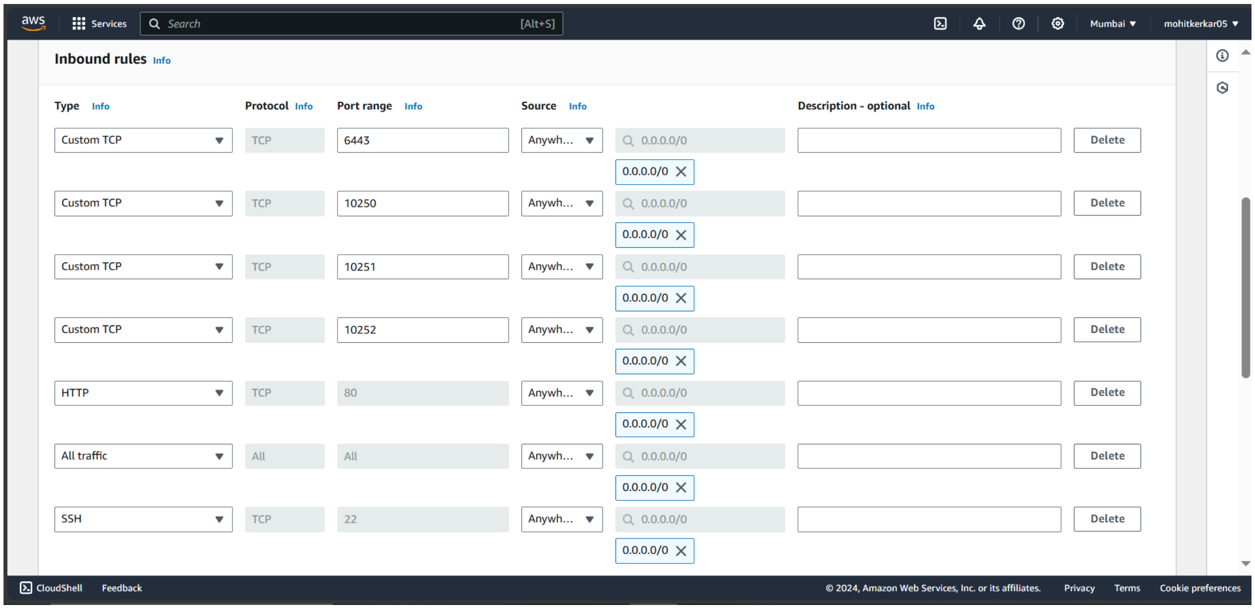
Security group name Info: mastersed

Name cannot be edited after creation.

Description Info: made for the master node

VPC Info: vpc-07f6fea081518a7f1 (vpcdefault)

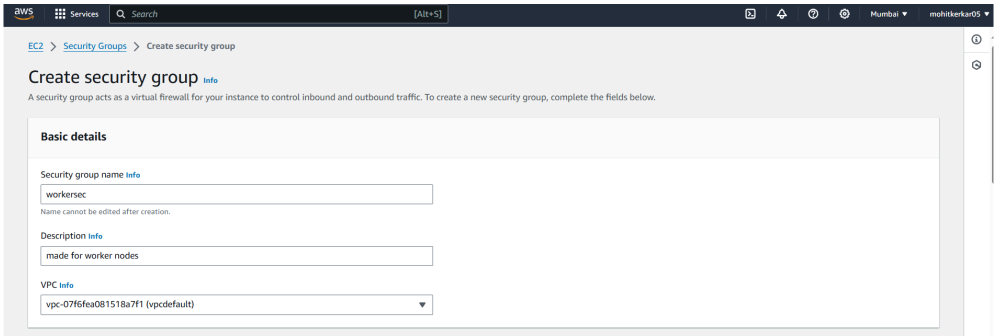
I modified the inbound rules in the following fashion for the master node



Inbound rules Info

Type Info	Protocol Info	Port range Info	Source Info		Description - optional Info	
Custom TCP	TCP	6443	Anywh...	0.0.0.0/0 0.0.0.0/0 ✕		Delete
Custom TCP	TCP	10250	Anywh...	0.0.0.0/0 0.0.0.0/0 ✕		Delete
Custom TCP	TCP	10251	Anywh...	0.0.0.0/0 0.0.0.0/0 ✕		Delete
Custom TCP	TCP	10252	Anywh...	0.0.0.0/0 0.0.0.0/0 ✕		Delete
HTTP	TCP	80	Anywh...	0.0.0.0/0 0.0.0.0/0 ✕		Delete
All traffic	All	All	Anywh...	0.0.0.0/0 0.0.0.0/0 ✕		Delete
SSH	TCP	22	Anywh...	0.0.0.0/0 0.0.0.0/0 ✕		Delete

Then, i created a security group for the worker nodes



EC2 > Security Groups > Create security group

Create security group Info

A security group acts as a virtual firewall for your instance to control inbound and outbound traffic. To create a new security group, complete the fields below.

Basic details

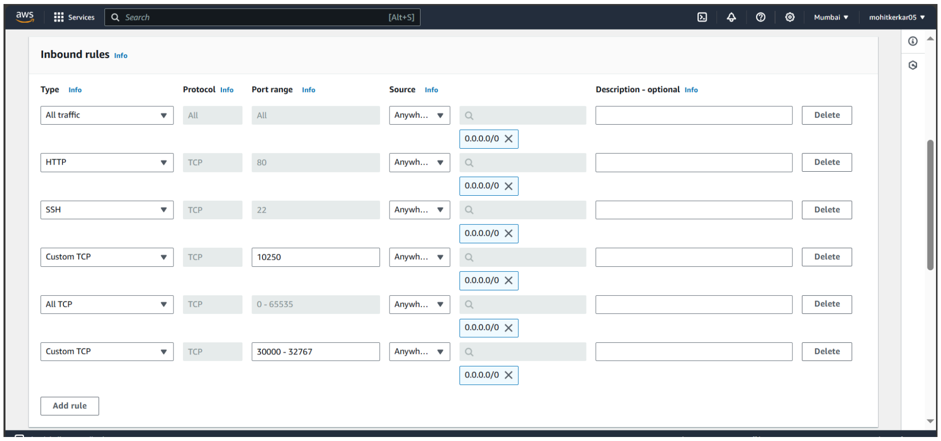
Security group name Info: workersec

Name cannot be edited after creation.

Description Info: made for worker nodes

VPC Info: vpc-07f6fea081518a7f1 (vpcdefault)

Edited the inbound rules for the worker nodes in the following fashion



Inbound rules Info

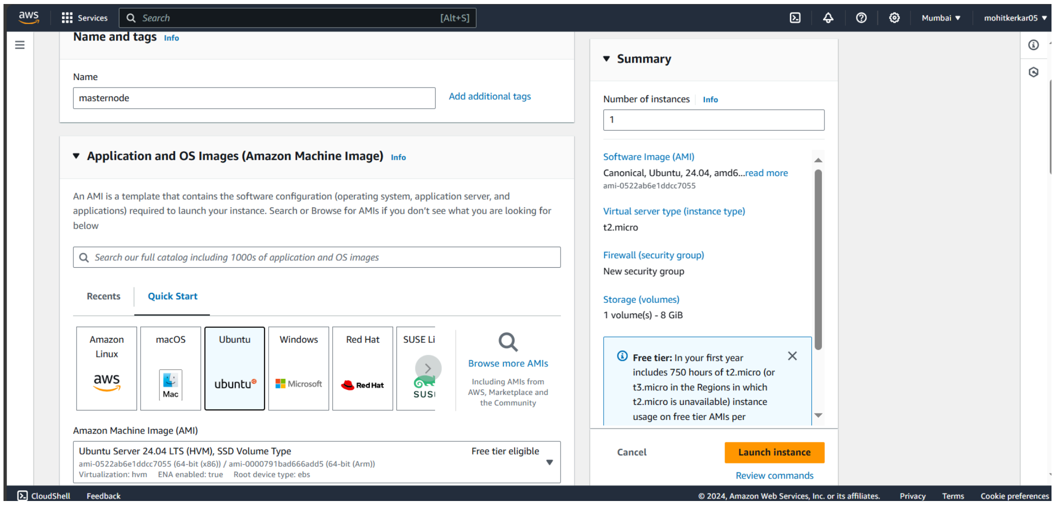
Type	Protocol	Port range	Source		Description - optional	
All traffic	All	All	Anywh...	0.0.0.0/0		Delete
HTTP	TCP	80	Anywh...	0.0.0.0/0		Delete
SSH	TCP	22	Anywh...	0.0.0.0/0		Delete
Custom TCP	TCP	10250	Anywh...	0.0.0.0/0		Delete
All TCP	TCP	0 - 65535	Anywh...	0.0.0.0/0		Delete
Custom TCP	TCP	30000 - 32767	Anywh...	0.0.0.0/0		Delete

Add rule

Step 2: Create 3 EC2 instances i.e one master and other 2 workers

Now, in order to work with the instances, navigate to EC2 instances section and launch a few instances.
Launch one instance and name it as masternode and the other two as worker.
I selected **AMI as ubuntu** among the list of OS that we shown available in the free tier eligibility list

Following is the master instance node:



Name and tags Info

Name: masternode

Add additional tags

Application and OS Images (Amazon Machine Image) Info

An AMI is a template that contains the software configuration (operating system, application server, and applications) required to launch your instance. Search or Browse for AMIs if you don't see what you are looking for below

Recents | Quick Start

Amazon Linux | macOS | Ubuntu | Windows | Red Hat | SUSE Li

Browse more AMIs — Including AMIs from AWS, Marketplace and the Community

Amazon Machine Image (AMI)

Ubuntu Server 24.04 LTS (HVM), SSD Volume Type — Free tier eligible
ami-0522ab6e1ddc7055 (64-bit (x86)) / ami-0000791bad666add5 (64-bit (Arm))
Virtualization: hvm ENA enabled: true Root device type: ebs

Summary

Number of instances Info: 1

Software Image (AMI): Canonical, Ubuntu, 24.04, amd6...read more
ami-0522ab6e1ddcc7055

Virtual server type (instance type): t2.micro

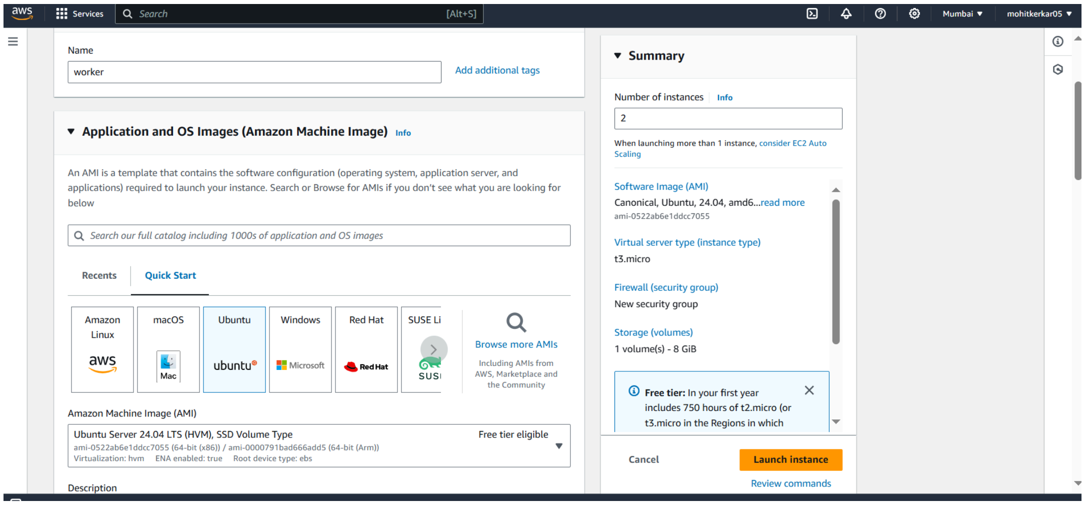
Firewall (security group): New security group

Storage (volumes): 1 volume(s) - 8 GiB

Free tier: In your first year includes 750 hours of t2.micro (or t3.micro in the Regions in which t2.micro is unavailable) instance usage on free tier AMIs per

Cancel | Launch instance | Review commands

Following is the worker instance node:



Name

worker

Add additional tags

▼ Application and OS Images (Amazon Machine Image) Info

An AMI is a template that contains the software configuration (operating system, application server, and applications) required to launch your instance. Search or Browse for AMIs if you don't see what you are looking for below

Search our full catalog including 1000s of application and OS images

Amazon Machine Image (AMI)

Ubuntu Server 24.04 LTS (HVM), SSD Volume Type Free tier eligible
ami-0522ab6e1ddcc7055 (64-bit (x86)) / ami-0000791bad666add5 (64-bit (Arm))
Virtualization: hvm ENA enabled: true Root device type: ebs

Description

▼ Summary

Number of instances Info

2

When launching more than 1 instance, consider EC2 Auto Scaling

Software Image (AMI)
Canonical, Ubuntu, 24.04, amd6...read more
ami-0522ab6e1ddcc7055

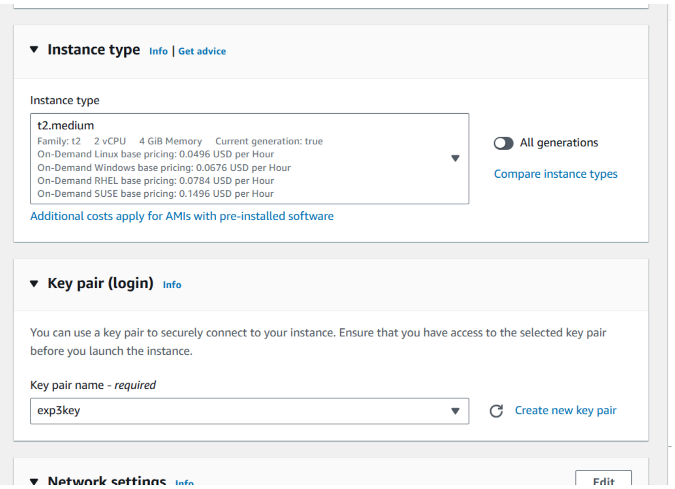
Virtual server type (instance type)
t3.micro

Firewall (security group)
New security group

Storage (volumes)
1 volume(s) - 8 GiB

Free tier: In your first year includes 750 hours of t2.micro (or t3.micro in the Regions in which

IMPORTANT: Select t2.medium as the instance type for all 3 instances, as kubernetes requires at least 2 CPUs to work with and sufficient amount of other resources as well


▼ Instance type Info | Get advice

Instance type

t2.medium
Family: t2 2 vCPU 4 GiB Memory Current generation: true
On-Demand Linux base pricing: 0.0496 USD per Hour
On-Demand Windows base pricing: 0.0676 USD per Hour
On-Demand RHEL base pricing: 0.0784 USD per Hour
On-Demand SUSE base pricing: 0.1496 USD per Hour

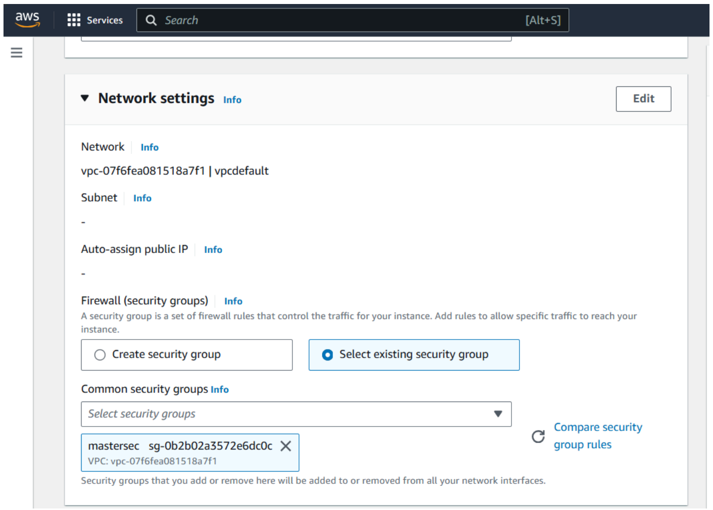
Additional costs apply for AMIs with pre-installed software

▼ Key pair (login) Info

You can use a key pair to securely connect to your instance. Ensure that you have access to the selected key pair before you launch the instance.

Key pair name - *required*

exp3key



▼ Network settings Info

Network Info
vpc-07f6fea081518a7f1 | vpcdefault

Subnet Info
-

Auto-assign public IP Info
-

Firewall (security groups) Info
A security group is a set of firewall rules that control the traffic for your instance. Add rules to allow specific traffic to reach your instance.

○ Create security group ● Select existing security group

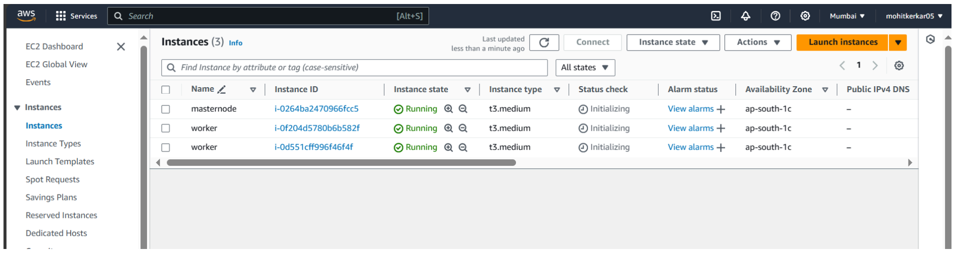
Common security groups Info

Select security groups

mastersec sg-0b2b02a3572e6dc0c
VPC: vpc-07f6fea081518a7f1

Security groups that you add or remove here will be added to or removed from all your network interfaces.

After launching all the 3 instances, select one instance and press ‘Connect’

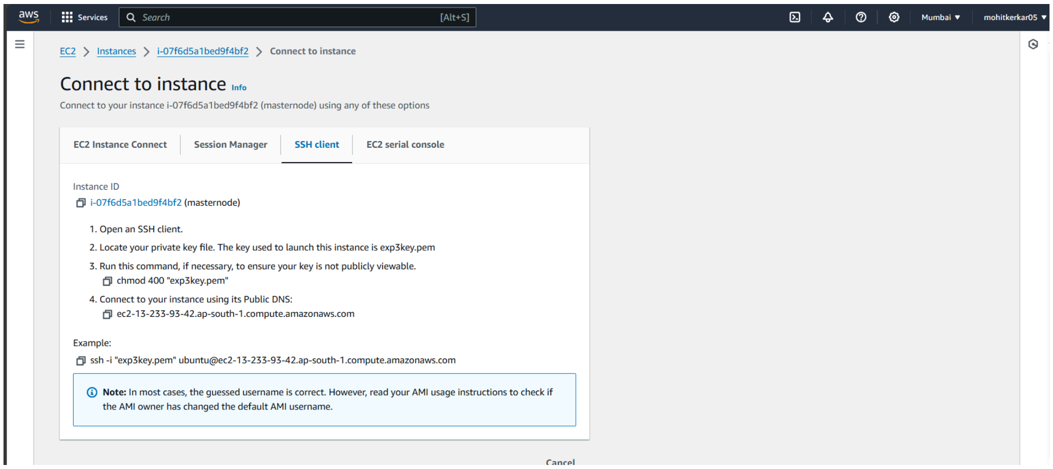


Instances (3) Info

Find Instance by attribute or tag (case-sensitive)

□	Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS
□	masternode	i-0264ba2470966fcc5	Running	t3.medium	Initializing	View alarms +	ap-south-1c	–
□	worker	i-0f204d5780b6b582f	Running	t3.medium	Initializing	View alarms +	ap-south-1c	–
□	worker	i-0d551cff996f46f4f	Running	t3.medium	Initializing	View alarms +	ap-south-1c	–

Navigate to SSH client section and copy the command thats listed for connecting to the node remotely
ssh -i "exp3key.pem" ubuntu@ec2-13-233-93-42.ap-south-1.compute.amazonaws.com…for the master node
Similarly, do this for all nodes



Connect to instance Info

Connect to your instance i-07f6d5a1bed9f4bf2 (masternode) using any of these options

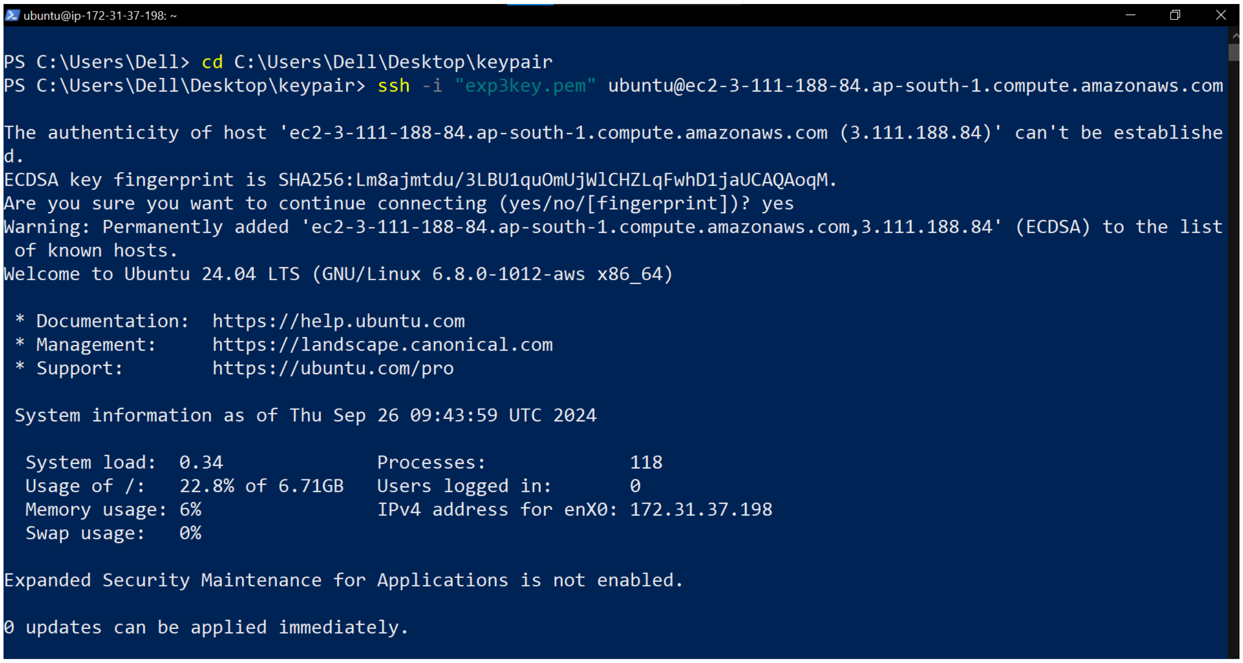
Instance ID
i-07f6d5a1bed9f4bf2 (masternode)

1. Open an SSH client.
2. Locate your private key file. The key used to launch this instance is exp3key.pem
3. Run this command, if necessary, to ensure your key is not publicly viewable.
   chmod 400 "exp3key.pem"
4. Connect to your instance using its Public DNS:
   ec2-13-233-93-42.ap-south-1.compute.amazonaws.com

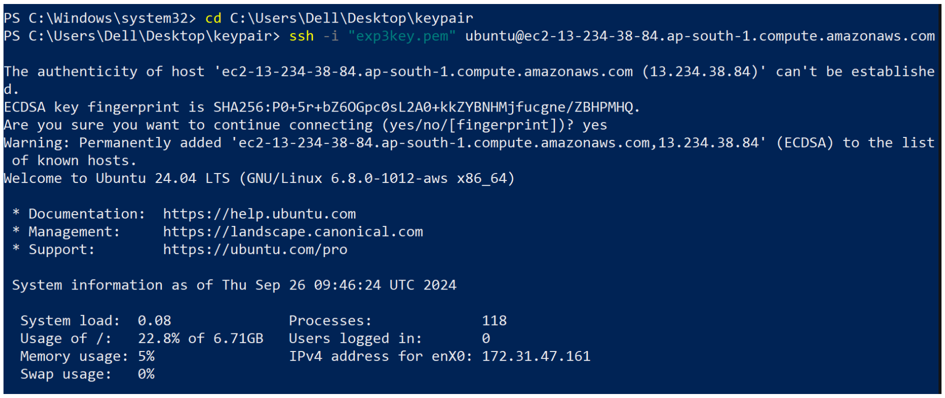
Example:
ssh -i "exp3key.pem" ubuntu@ec2-13-233-93-42.ap-south-1.compute.amazonaws.com

Note: In most cases, the guessed username is correct. However, read your AMI usage instructions to check if the AMI owner has changed the default AMI username.

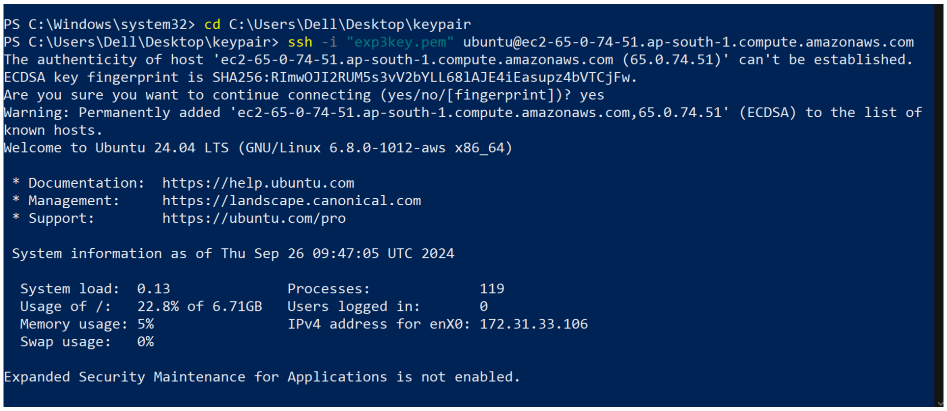
Now, open 3 separate terminals and run change directories to the folder which contains the key we created earlier
Run their respective SSH commands (one in each terminal)
This helps us log onto those instances individually and remotely and work on them separately



```
PS C:\Users\Dell> cd C:\Users\Dell\Desktop\keypair
PS C:\Users\Dell\Desktop\keypair> ssh -i "exp3key.pem" ubuntu@ec2-3-111-188-84.ap-south-1.compute.amazonaws.com

The authenticity of host 'ec2-3-111-188-84.ap-south-1.compute.amazonaws.com (3.111.188.84)' can't be establishe
d.
ECDSA key fingerprint is SHA256:Lm8ajmtdu/3LBU1quOmUjWlCHZLqFwhD1jaUCAQAoqM.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'ec2-3-111-188-84.ap-south-1.compute.amazonaws.com,3.111.188.84' (ECDSA) to the list
 of known hosts.
Welcome to Ubuntu 24.04 LTS (GNU/Linux 6.8.0-1012-aws x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/pro

 System information as of Thu Sep 26 09:43:59 UTC 2024

  System load:  0.34               Processes:             118
  Usage of /:   22.8% of 6.71GB    Users logged in:       0
  Memory usage: 6%                 IPv4 address for enX0: 172.31.37.198
  Swap usage:   0%

Expanded Security Maintenance for Applications is not enabled.

0 updates can be applied immediately.
```



```
PS C:\Windows\system32> cd C:\Users\Dell\Desktop\keypair
PS C:\Users\Dell\Desktop\keypair> ssh -i "exp3key.pem" ubuntu@ec2-13-234-38-84.ap-south-1.compute.amazonaws.com

The authenticity of host 'ec2-13-234-38-84.ap-south-1.compute.amazonaws.com (13.234.38.84)' can't be establishe
d.
ECDSA key fingerprint is SHA256:P0+5r+bZ6OGpc0sL2A0+kkZYBNHMjfucgne/ZBHPMHQ.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'ec2-13-234-38-84.ap-south-1.compute.amazonaws.com,13.234.38.84' (ECDSA) to the list
 of known hosts.
Welcome to Ubuntu 24.04 LTS (GNU/Linux 6.8.0-1012-aws x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/pro

 System information as of Thu Sep 26 09:46:24 UTC 2024

  System load:  0.08              Processes:             118
  Usage of /:   22.8% of 6.71GB   Users logged in:       0
  Memory usage: 5%                IPv4 address for enX0: 172.31.47.161
  Swap usage:   0%
```

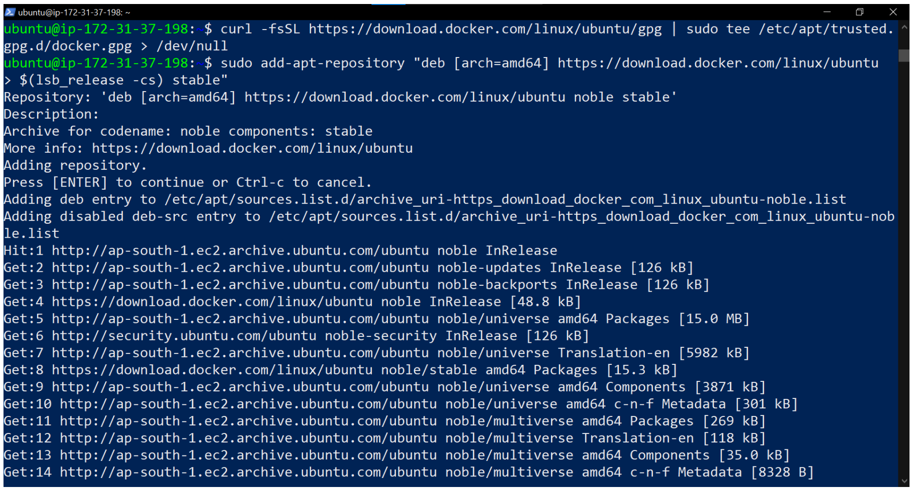


```
PS C:\Windows\system32> cd C:\Users\Dell\Desktop\keypair
PS C:\Users\Dell\Desktop\keypair> ssh -i "exp3key.pem" ubuntu@ec2-65-0-74-51.ap-south-1.compute.amazonaws.com
The authenticity of host 'ec2-65-0-74-51.ap-south-1.compute.amazonaws.com (65.0.74.51)' can't be established.
ECDSA key fingerprint is SHA256:RImwOJI2RUM5s3vV2bYLL68lAJE4iEasupz4bVTCjFw.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'ec2-65-0-74-51.ap-south-1.compute.amazonaws.com,65.0.74.51' (ECDSA) to the list of
known hosts.
Welcome to Ubuntu 24.04 LTS (GNU/Linux 6.8.0-1012-aws x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/pro

 System information as of Thu Sep 26 09:47:05 UTC 2024

  System load:  0.13              Processes:             119
  Usage of /:   22.8% of 6.71GB   Users logged in:       0
  Memory usage: 5%                IPv4 address for enX0: 172.31.33.106
  Swap usage:   0%

Expanded Security Maintenance for Applications is not enabled.
```

Step 3: Docker installation:

Run the following command: These commands are used to install Docker on an Ubuntu system by adding Docker's official GPG key and configuring the Docker repository.

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo tee /etc/apt/trusted.gpg.d/docker.gpg > /dev/null
sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"

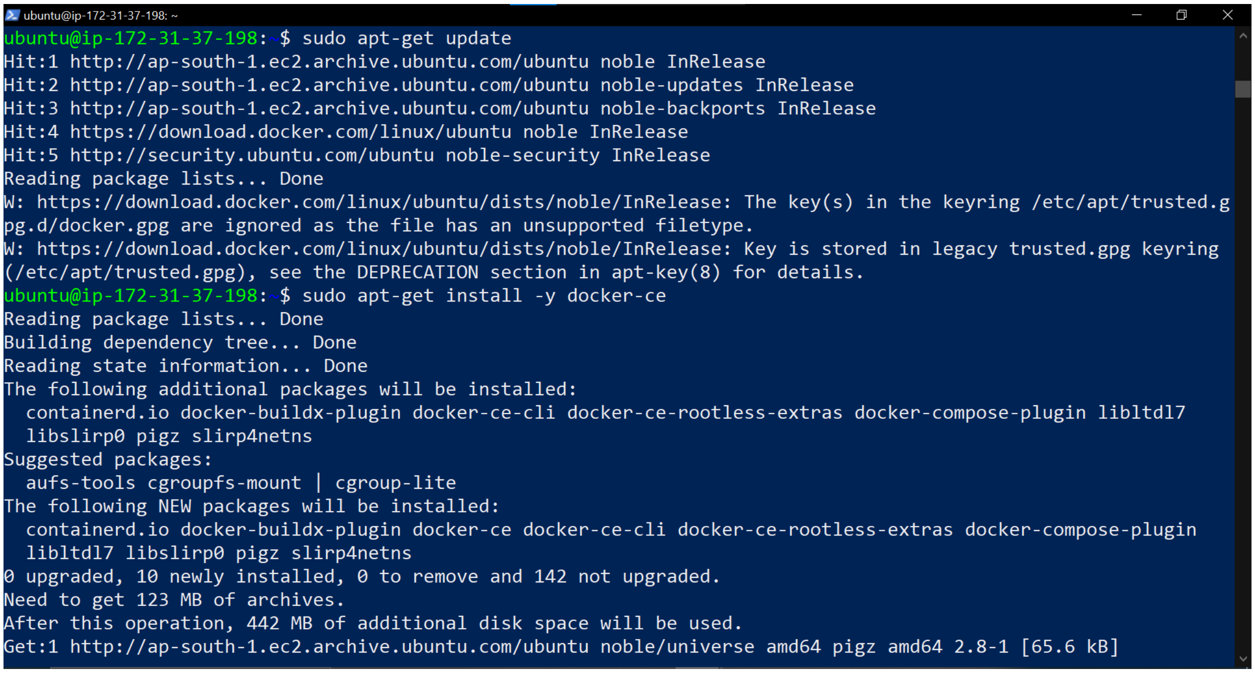


```
ubuntu@ip-172-31-37-198:~$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo tee /etc/apt/trusted.
gpg.d/docker.gpg > /dev/null
ubuntu@ip-172-31-37-198:~$ sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu
> $(lsb_release -cs) stable"
Repository: 'deb [arch=amd64] https://download.docker.com/linux/ubuntu noble stable'
Description:
Archive for codename: noble components: stable
More info: https://download.docker.com/linux/ubuntu
Adding repository.
Press [ENTER] to continue or Ctrl-c to cancel.
Adding deb entry to /etc/apt/sources.list.d/archive_uri-https_download_docker_com_linux_ubuntu-noble.list
Adding disabled deb-src entry to /etc/apt/sources.list.d/archive_uri-https_download_docker_com_linux_ubuntu-nob
le.list
Hit:1 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu noble InRelease
Get:2 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu noble-updates InRelease [126 kB]
Get:3 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu noble-backports InRelease [126 kB]
Get:4 https://download.docker.com/linux/ubuntu noble InRelease [48.8 kB]
Get:5 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu noble/universe amd64 Packages [15.0 MB]
Get:6 http://security.ubuntu.com/ubuntu noble-security InRelease [126 kB]
Get:7 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu noble/universe Translation-en [5982 kB]
Get:8 https://download.docker.com/linux/ubuntu noble/stable amd64 Packages [15.3 kB]
Get:9 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu noble/universe amd64 Components [3871 kB]
Get:10 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu noble/universe amd64 c-n-f Metadata [301 kB]
Get:11 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu noble/multiverse amd64 Packages [269 kB]
Get:12 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu noble/multiverse Translation-en [118 kB]
Get:13 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu noble/multiverse amd64 Components [35.0 kB]
Get:14 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu noble/multiverse amd64 c-n-f Metadata [8328 B]
```

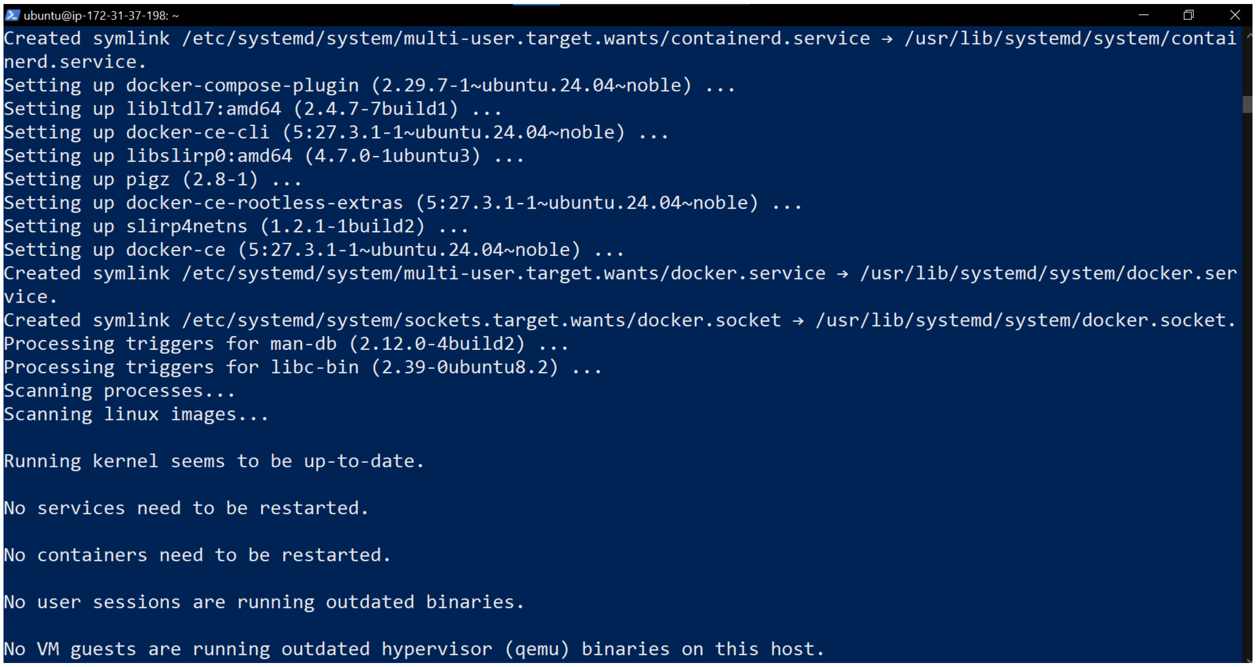
Name: Mohit Kerkar Div: D15C Roll No: 23

Run the following commands to refresh your local package list to ensure the latest packages are available and install Docker Community Edition on your system without prompting for confirmation.

sudo apt-get update
sudo apt-get install -y docker-ce



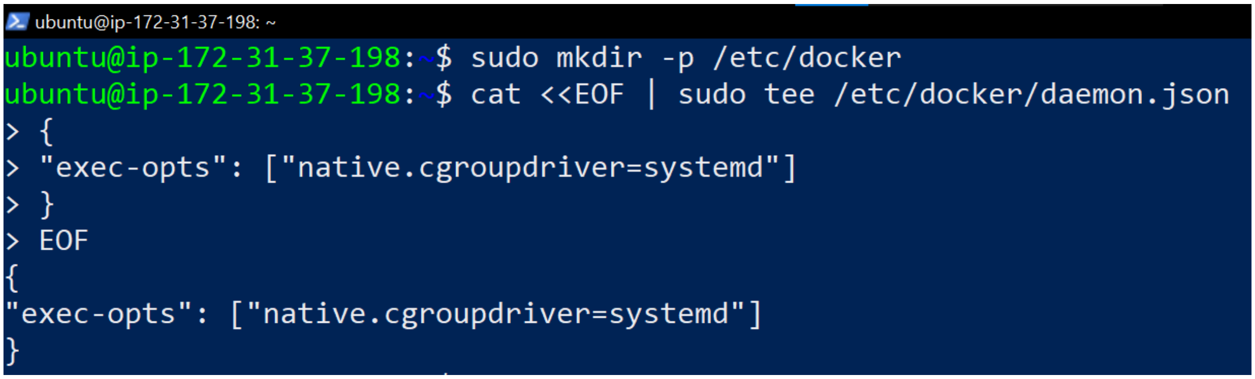
```
ubuntu@ip-172-31-37-198:~$ sudo apt-get update
Hit:1 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu noble InRelease
Hit:2 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu noble-updates InRelease
Hit:3 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu noble-backports InRelease
Hit:4 https://download.docker.com/linux/ubuntu noble InRelease
Hit:5 http://security.ubuntu.com/ubuntu noble-security InRelease
Reading package lists... Done
W: https://download.docker.com/linux/ubuntu/dists/noble/InRelease: The key(s) in the keyring /etc/apt/trusted.g
pg.d/docker.gpg are ignored as the file has an unsupported filetype.
W: https://download.docker.com/linux/ubuntu/dists/noble/InRelease: Key is stored in legacy trusted.gpg keyring
(/etc/apt/trusted.gpg), see the DEPRECATION section in apt-key(8) for details.
ubuntu@ip-172-31-37-198:~$ sudo apt-get install -y docker-ce
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  containerd.io docker-buildx-plugin docker-ce-cli docker-ce-rootless-extras docker-compose-plugin libltdl7
  libslirp0 pigz slirp4netns
Suggested packages:
  aufs-tools cgroupfs-mount | cgroup-lite
The following NEW packages will be installed:
  containerd.io docker-buildx-plugin docker-ce docker-ce-cli docker-ce-rootless-extras docker-compose-plugin
  libltdl7 libslirp0 pigz slirp4netns
0 upgraded, 10 newly installed, 0 to remove and 142 not upgraded.
Need to get 123 MB of archives.
After this operation, 442 MB of additional disk space will be used.
Get:1 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu noble/universe amd64 pigz amd64 2.8-1 [65.6 kB]
```



```
Created symlink /etc/systemd/system/multi-user.target.wants/containerd.service → /usr/lib/systemd/system/contai
nerd.service.
Setting up docker-compose-plugin (2.29.7-1~ubuntu.24.04~noble) ...
Setting up libltdl7:amd64 (2.4.7-7build1) ...
Setting up docker-ce-cli (5:27.3.1-1~ubuntu.24.04~noble) ...
Setting up libslirp0:amd64 (4.7.0-1ubuntu3) ...
Setting up pigz (2.8-1) ...
Setting up docker-ce-rootless-extras (5:27.3.1-1~ubuntu.24.04~noble) ...
Setting up slirp4netns (1.2.1-1build2) ...
Setting up docker-ce (5:27.3.1-1~ubuntu.24.04~noble) ...
Created symlink /etc/systemd/system/multi-user.target.wants/docker.service → /usr/lib/systemd/system/docker.ser
vice.
Created symlink /etc/systemd/system/sockets.target.wants/docker.socket → /usr/lib/systemd/system/docker.socket.
Processing triggers for man-db (2.12.0-4build2) ...
Processing triggers for libc-bin (2.39-0ubuntu8.2) ...
Scanning processes...
Scanning linux images...

Running kernel seems to be up-to-date.

No services need to be restarted.

No containers need to be restarted.

No user sessions are running outdated binaries.

No VM guests are running outdated hypervisor (qemu) binaries on this host.
```

Run the following commands to create the Docker configuration directory and write a configuration file for Docker to use the systemd cgroup driver.

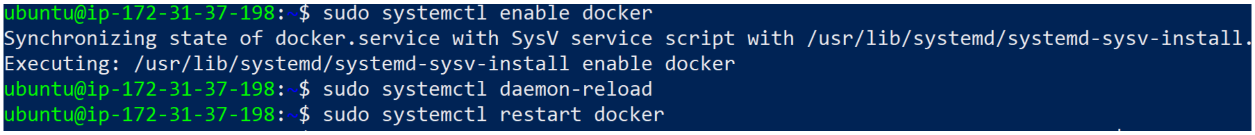
```
sudo mkdir -p /etc/docker
cat <<EOF | sudo tee /etc/docker/daemon.json
{
"exec-opts": ["native.cgroupdriver=systemd"]
}
EOF
```



```
ubuntu@ip-172-31-37-198:~$ sudo mkdir -p /etc/docker
ubuntu@ip-172-31-37-198:~$ cat <<EOF | sudo tee /etc/docker/daemon.json
> {
> "exec-opts": ["native.cgroupdriver=systemd"]
> }
> EOF
{
"exec-opts": ["native.cgroupdriver=systemd"]
}
```

Run the following commands to ensure Docker starts automatically on system boot, reload systemd to apply new configurations, restart Docker to apply the new configuration.

```
sudo systemctl enable docker
sudo systemctl daemon-reload
sudo systemctl restart docker
```

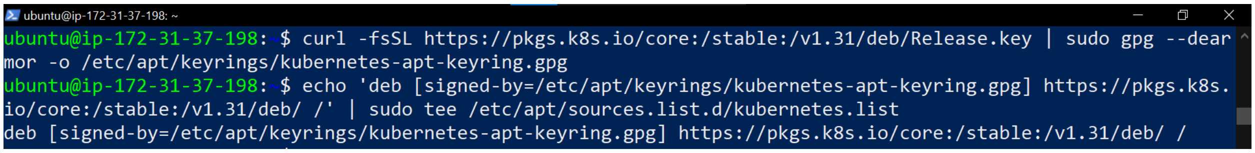


```
ubuntu@ip-172-31-37-198:~$ sudo systemctl enable docker
Synchronizing state of docker.service with SysV service script with /usr/lib/systemd/systemd-sysv-install.
Executing: /usr/lib/systemd/systemd-sysv-install enable docker
ubuntu@ip-172-31-37-198:~$ sudo systemctl daemon-reload
ubuntu@ip-172-31-37-198:~$ sudo systemctl restart docker
```

Step 4: Kubernetes Installation:

Run the following commands to add the Kubernetes package repository to your Ubuntu system in order to install Kubernetes components like kubectl, kubelet, and kubeadm.

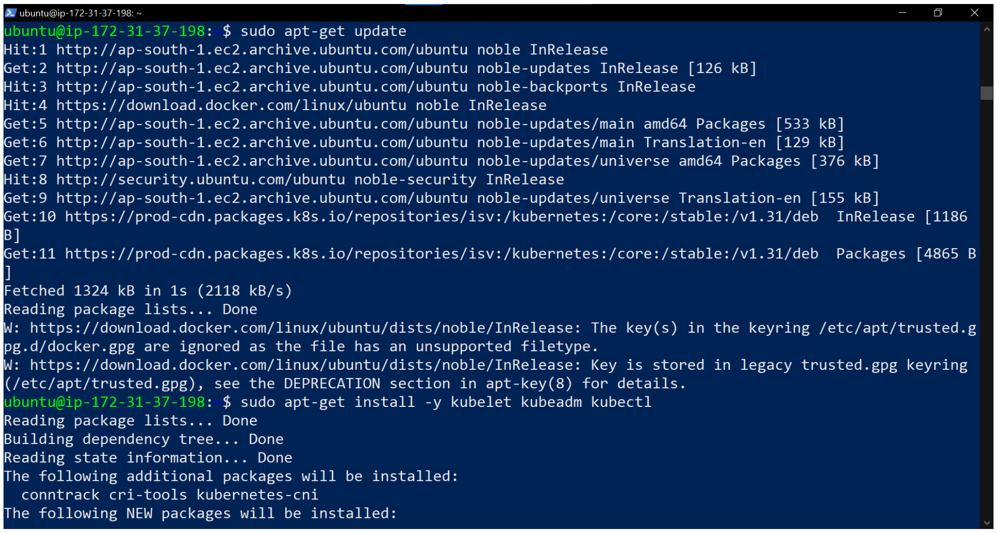
```
curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.31/deb/Release.key | sudo gpg --dearmor -o
/etc/apt/keyrings/kubernetes-apt-keyring.gpg
echo 'deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg]
https://pkgs.k8s.io/core:/stable:/v1.31/deb/ /' | sudo tee /etc/apt/sources.list.d/kubernetes.list
```



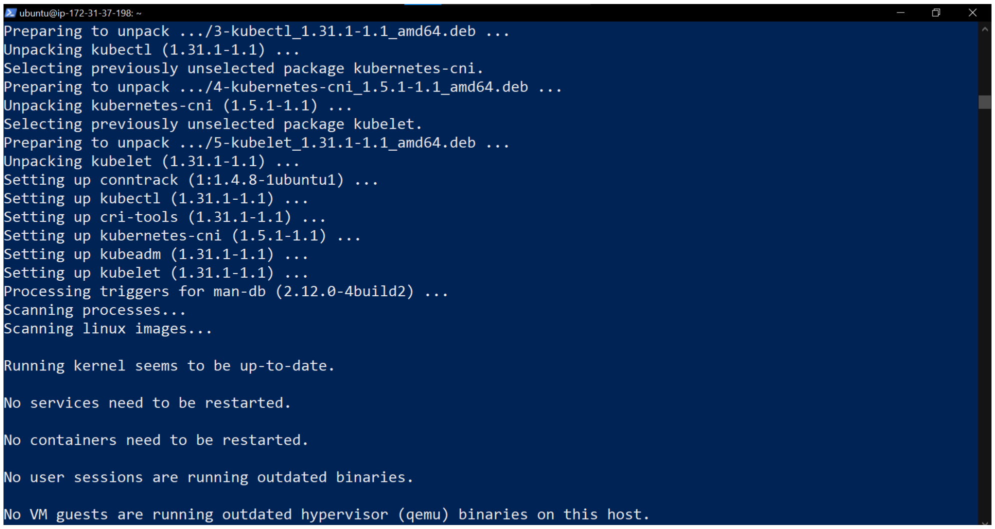
```
ubuntu@ip-172-31-37-198:~$ curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.31/deb/Release.key | sudo gpg --dear
mor -o /etc/apt/keyrings/kubernetes-apt-keyring.gpg
ubuntu@ip-172-31-37-198:~$ echo 'deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg] https://pkgs.k8s.
io/core:/stable:/v1.31/deb/ /' | sudo tee /etc/apt/sources.list.d/kubernetes.list
deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg] https://pkgs.k8s.io/core:/stable:/v1.31/deb/ /
```

Next, run these commands to refresh the local package index to include the newly added Kubernetes repository, install the main Kubernetes components (kubelet, kubeadm, kubectl), prevent the installed Kubernetes components from being automatically updated, ensuring cluster stability until manual updates are performed.

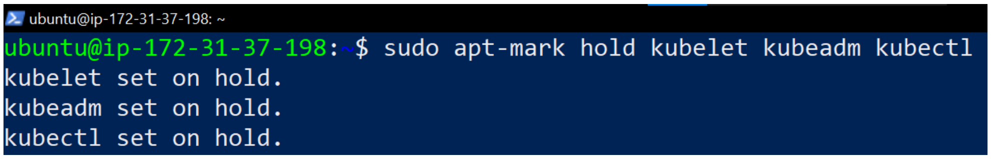
sudo apt-get update
sudo apt-get install -y kubelet kubeadm kubectl
sudo apt-mark hold kubelet kubeadm kubectl



```
ubuntu@ip-172-31-37-198:~$ sudo apt-get update
Hit:1 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu noble InRelease
Get:2 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu noble-updates InRelease [126 kB]
Hit:3 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu noble-backports InRelease
Hit:4 https://download.docker.com/linux/ubuntu noble InRelease
Get:5 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu noble-updates/main amd64 Packages [533 kB]
Get:6 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu noble-updates/main Translation-en [129 kB]
Get:7 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu noble-updates/universe amd64 Packages [376 kB]
Hit:8 http://security.ubuntu.com/ubuntu noble-security InRelease
Get:9 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu noble-updates/universe Translation-en [155 kB]
Get:10 https://prod-cdn.packages.k8s.io/repositories/isv:/kubernetes:/core:/stable:/v1.31/deb  InRelease [1186
B]
Get:11 https://prod-cdn.packages.k8s.io/repositories/isv:/kubernetes:/core:/stable:/v1.31/deb  Packages [4865 B
]
Fetched 1324 kB in 1s (2118 kB/s)
Reading package lists... Done
W: https://download.docker.com/linux/ubuntu/dists/noble/InRelease: The key(s) in the keyring /etc/apt/trusted.g
pg.d/docker.gpg are ignored as the file has an unsupported filetype.
W: https://download.docker.com/linux/ubuntu/dists/noble/InRelease: Key is stored in legacy trusted.gpg keyring
(/etc/apt/trusted.gpg), see the DEPRECATION section in apt-key(8) for details.
ubuntu@ip-172-31-37-198:~$ sudo apt-get install -y kubelet kubeadm kubectl
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  conntrack cri-tools kubernetes-cni
The following NEW packages will be installed:
```



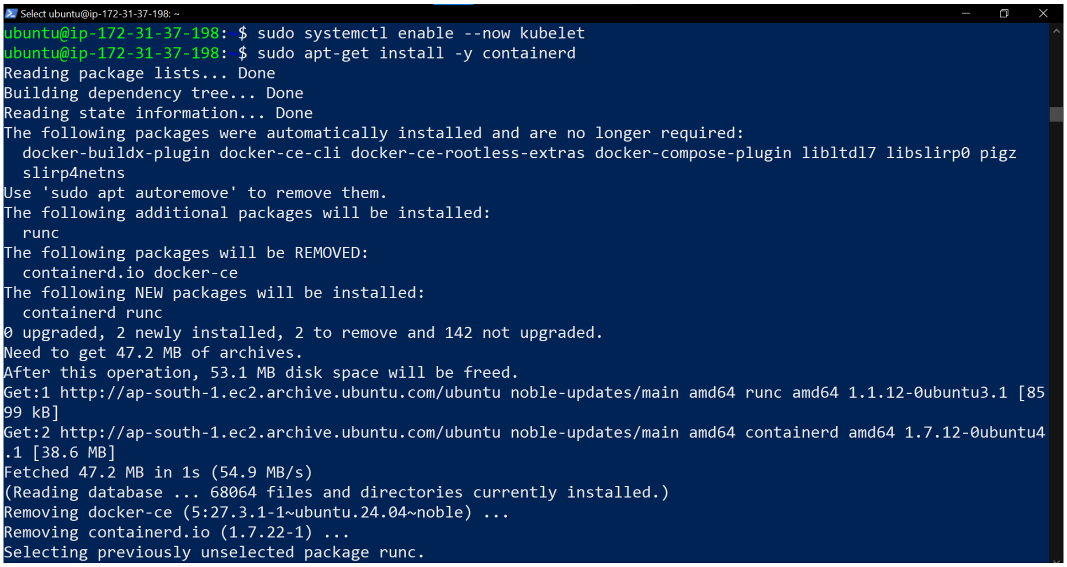
```
Preparing to unpack .../3-kubectl_1.31.1-1.1_amd64.deb ...
Unpacking kubectl (1.31.1-1.1) ...
Selecting previously unselected package kubernetes-cni.
Preparing to unpack .../4-kubernetes-cni_1.5.1-1.1_amd64.deb ...
Unpacking kubernetes-cni (1.5.1-1.1) ...
Selecting previously unselected package kubelet.
Preparing to unpack .../5-kubelet_1.31.1-1.1_amd64.deb ...
Unpacking kubelet (1.31.1-1.1) ...
Setting up conntrack (1:1.4.8-1ubuntu1) ...
Setting up kubectl (1.31.1-1.1) ...
Setting up cri-tools (1.31.1-1.1) ...
Setting up kubernetes-cni (1.5.1-1.1) ...
Setting up kubeadm (1.31.1-1.1) ...
Setting up kubelet (1.31.1-1.1) ...
Processing triggers for man-db (2.12.0-4build2) ...
Scanning processes...
Scanning linux images...

Running kernel seems to be up-to-date.

No services need to be restarted.

No containers need to be restarted.

No user sessions are running outdated binaries.

No VM guests are running outdated hypervisor (qemu) binaries on this host.
```



```
ubuntu@ip-172-31-37-198:~$ sudo apt-mark hold kubelet kubeadm kubectl
kubelet set on hold.
kubeadm set on hold.
kubectl set on hold.
```

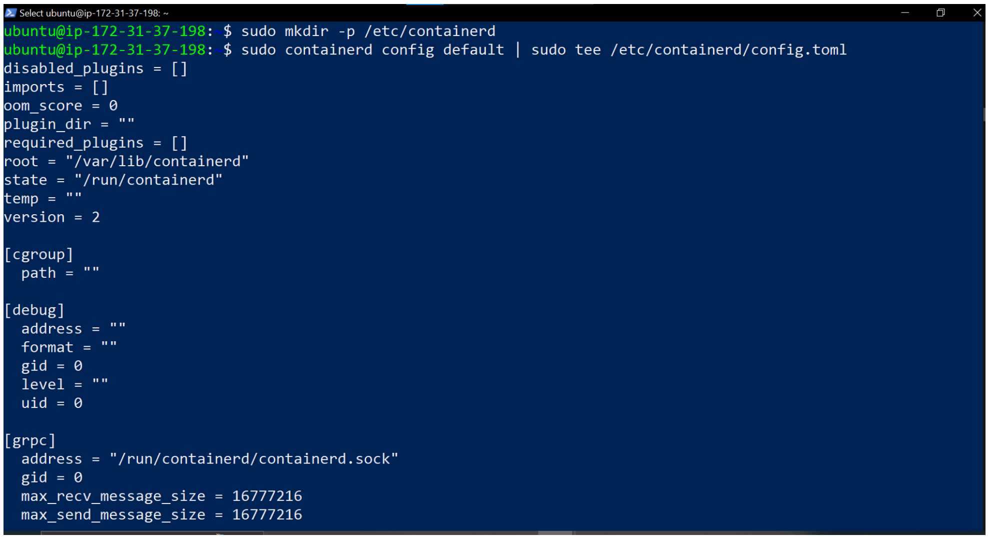
The command **sudo systemctl enable --now kubelet** enables the **kubelet** service, ensuring it starts automatically at boot and immediately starts running without needing a system reboot. The **sudo apt-get install -y containerd** command installs **containerd**, a container runtime that Kubernetes uses to manage and run containers.

sudo systemctl enable --now kubelet
sudo apt-get install -y containerd



```
ubuntu@ip-172-31-37-198:~$ sudo systemctl enable --now kubelet
ubuntu@ip-172-31-37-198:~$ sudo apt-get install -y containerd
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following packages were automatically installed and are no longer required:
  docker-buildx-plugin docker-ce-cli docker-ce-rootless-extras docker-compose-plugin libltdl7 libslirp0 pigz
  slirp4netns
Use 'sudo apt autoremove' to remove them.
The following additional packages will be installed:
  runc
The following packages will be REMOVED:
  containerd.io docker-ce
The following NEW packages will be installed:
  containerd runc
0 upgraded, 2 newly installed, 2 to remove and 142 not upgraded.
Need to get 47.2 MB of archives.
After this operation, 53.1 MB disk space will be freed.
Get:1 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu noble-updates/main amd64 runc amd64 1.1.12-0ubuntu3.1 [85
99 kB]
Get:2 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu noble-updates/main amd64 containerd amd64 1.7.12-0ubuntu4
.1 [38.6 MB]
Fetched 47.2 MB in 1s (54.9 MB/s)
(Reading database ... 68064 files and directories currently installed.)
Removing docker-ce (5:27.3.1-1~ubuntu.24.04~noble) ...
Removing containerd.io (1.7.22-1) ...
Selecting previously unselected package runc.
```

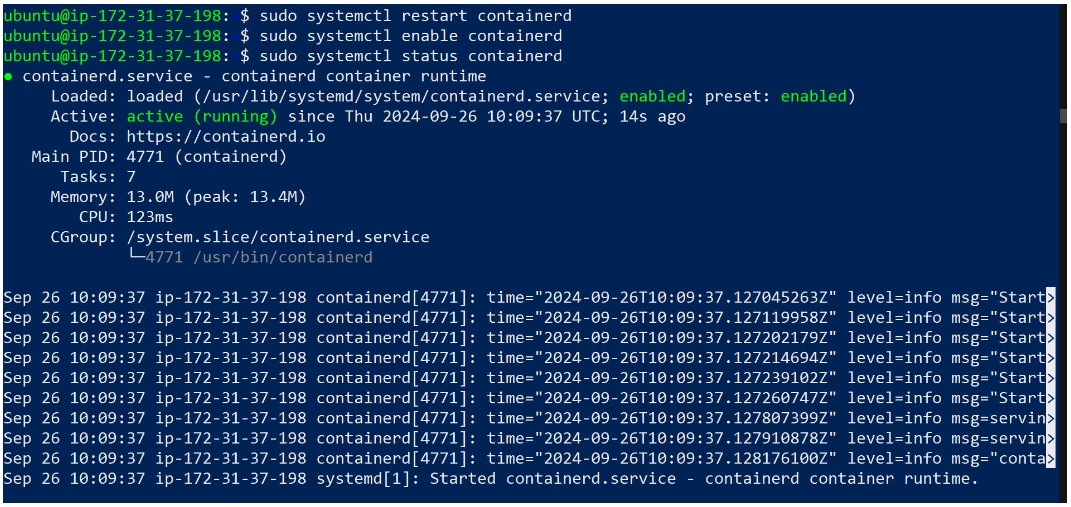
The command `sudo mkdir -p /etc/containerd` creates the directory for containerd configuration files if it doesn't already exist. The next command, `sudo containerd config default | sudo tee /etc/containerd/config.toml`, generates a default configuration for containerd and saves it as `config.toml` in that directory. Together, these commands set up the necessary directory and create a default configuration file for containerd.
sudo mkdir -p /etc/containerd
sudo containerd config default | sudo tee /etc/containerd/config.toml



```
ubuntu@ip-172-31-37-198:~$ sudo mkdir -p /etc/containerd
ubuntu@ip-172-31-37-198:~$ sudo containerd config default | sudo tee /etc/containerd/config.toml
disabled_plugins = []
imports = []
oom_score = 0
plugin_dir = ""
required_plugins = []
root = "/var/lib/containerd"
state = "/run/containerd"
temp = ""
version = 2

[cgroup]
  path = ""

[debug]
  address = ""
  format = ""
  gid = 0
  level = ""
  uid = 0

[grpc]
  address = "/run/containerd/containerd.sock"
  gid = 0
  max_recv_message_size = 16777216
  max_send_message_size = 16777216
```

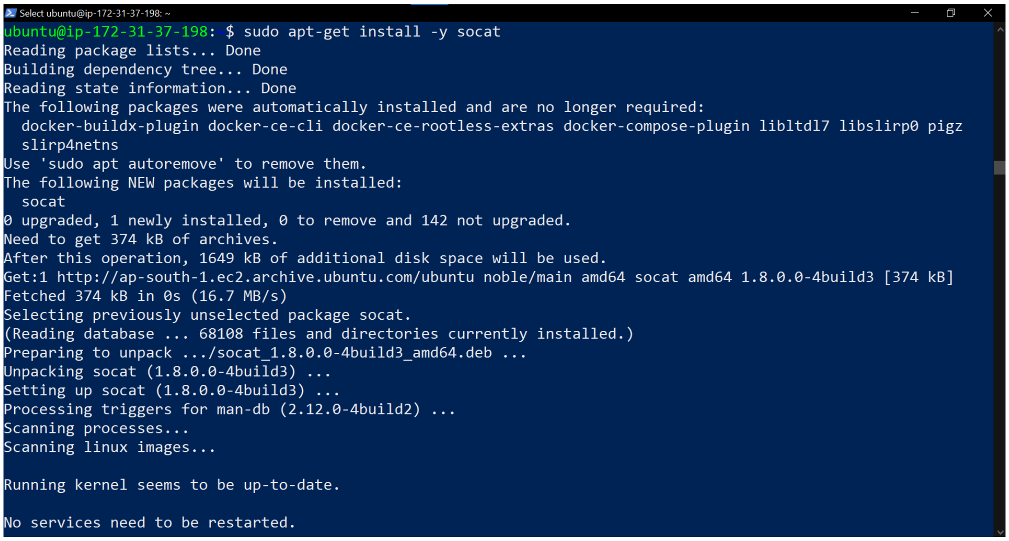
The command `sudo systemctl restart containerd` restarts the `containerd` service to apply any changes. `sudo systemctl enable containerd` sets it to start automatically at boot, while `sudo systemctl status containerd` checks its current status, showing whether it's active and any errors. Together, these commands manage the operation and health of the container runtime.

sudo systemctl restart containerd
sudo systemctl enable containerd
sudo systemctl status containerd



```
ubuntu@ip-172-31-37-198:~$ sudo systemctl restart containerd
ubuntu@ip-172-31-37-198:~$ sudo systemctl enable containerd
ubuntu@ip-172-31-37-198:~$ sudo systemctl status containerd
● containerd.service - containerd container runtime
     Loaded: loaded (/usr/lib/systemd/system/containerd.service; enabled; preset: enabled)
     Active: active (running) since Thu 2024-09-26 10:09:37 UTC; 14s ago
       Docs: https://containerd.io
   Main PID: 4771 (containerd)
      Tasks: 7
     Memory: 13.0M (peak: 13.4M)
        CPU: 123ms
     CGroup: /system.slice/containerd.service
             └─4771 /usr/bin/containerd

Sep 26 10:09:37 ip-172-31-37-198 containerd[4771]: time="2024-09-26T10:09:37.127045263Z" level=info msg="Start>
Sep 26 10:09:37 ip-172-31-37-198 containerd[4771]: time="2024-09-26T10:09:37.127119958Z" level=info msg="Start>
Sep 26 10:09:37 ip-172-31-37-198 containerd[4771]: time="2024-09-26T10:09:37.127202179Z" level=info msg="Start>
Sep 26 10:09:37 ip-172-31-37-198 containerd[4771]: time="2024-09-26T10:09:37.127214694Z" level=info msg="Start>
Sep 26 10:09:37 ip-172-31-37-198 containerd[4771]: time="2024-09-26T10:09:37.127239102Z" level=info msg="Start>
Sep 26 10:09:37 ip-172-31-37-198 containerd[4771]: time="2024-09-26T10:09:37.127260747Z" level=info msg="Start>
Sep 26 10:09:37 ip-172-31-37-198 containerd[4771]: time="2024-09-26T10:09:37.127807399Z" level=info msg=servin>
Sep 26 10:09:37 ip-172-31-37-198 containerd[4771]: time="2024-09-26T10:09:37.127910878Z" level=info msg=servin>
Sep 26 10:09:37 ip-172-31-37-198 containerd[4771]: time="2024-09-26T10:09:37.128176100Z" level=info msg="conta>
Sep 26 10:09:37 ip-172-31-37-198 systemd[1]: Started containerd.service - containerd container runtime.
```

Socat installation:
sudo apt-get install -y socat



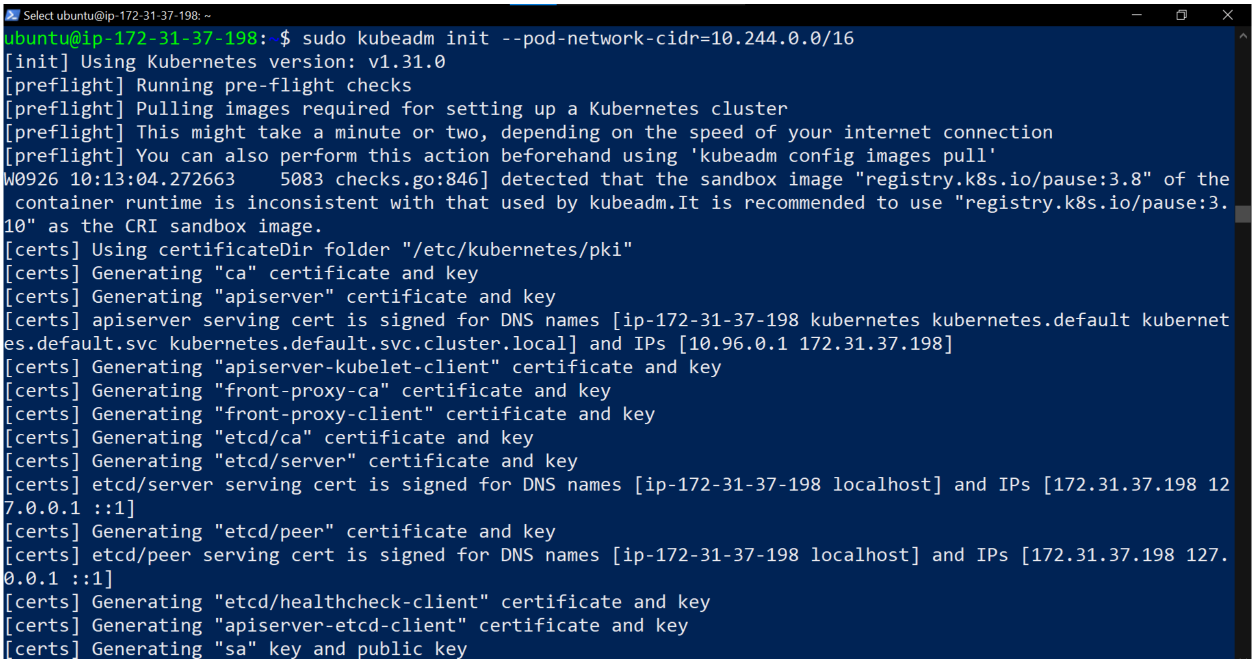
```
ubuntu@ip-172-31-37-198:~$ sudo apt-get install -y socat
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following packages were automatically installed and are no longer required:
  docker-buildx-plugin docker-ce-cli docker-ce-rootless-extras docker-compose-plugin libltdl7 libslirp0 pigz
  slirp4netns
Use 'sudo apt autoremove' to remove them.
The following NEW packages will be installed:
  socat
0 upgraded, 1 newly installed, 0 to remove and 142 not upgraded.
Need to get 374 kB of archives.
After this operation, 1649 kB of additional disk space will be used.
Get:1 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu noble/main amd64 socat amd64 1.8.0.0-4build3 [374 kB]
Fetched 374 kB in 0s (16.7 MB/s)
Selecting previously unselected package socat.
(Reading database ... 68108 files and directories currently installed.)
Preparing to unpack .../socat_1.8.0.0-4build3_amd64.deb ...
Unpacking socat (1.8.0.0-4build3) ...
Setting up socat (1.8.0.0-4build3) ...
Processing triggers for man-db (2.12.0-4build2) ...
Scanning processes...
Scanning linux images...

Running kernel seems to be up-to-date.

No services need to be restarted.
```

Step 5: Kubernetes Cluster

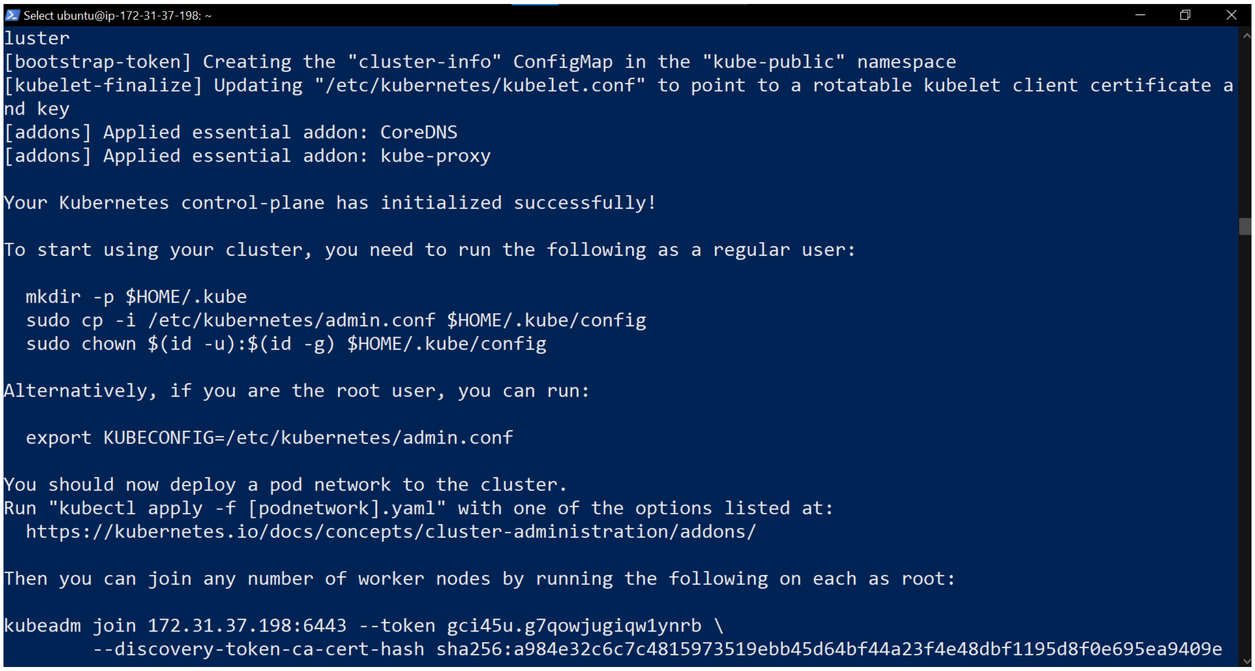
Run this command only on the master instance

sudo kubeadm init --pod-network-cidr=10.244.0.0/16

It initializes a Kubernetes cluster with kubeadm and sets up the control plane (master node).

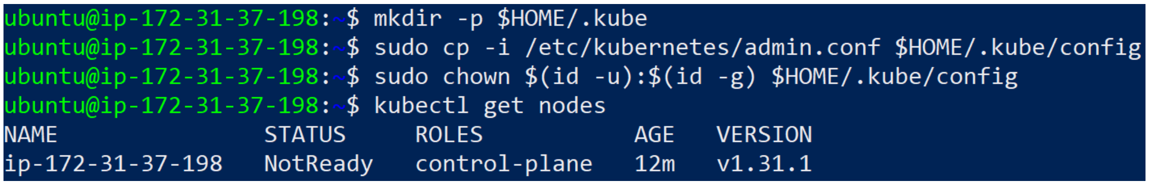


```
ubuntu@ip-172-31-37-198:~$ sudo kubeadm init --pod-network-cidr=10.244.0.0/16
[init] Using Kubernetes version: v1.31.0
[preflight] Running pre-flight checks
[preflight] Pulling images required for setting up a Kubernetes cluster
[preflight] This might take a minute or two, depending on the speed of your internet connection
[preflight] You can also perform this action beforehand using 'kubeadm config images pull'
W0926 10:13:04.272663    5083 checks.go:846] detected that the sandbox image "registry.k8s.io/pause:3.8" of the
 container runtime is inconsistent with that used by kubeadm.It is recommended to use "registry.k8s.io/pause:3.
10" as the CRI sandbox image.
[certs] Using certificateDir folder "/etc/kubernetes/pki"
[certs] Generating "ca" certificate and key
[certs] Generating "apiserver" certificate and key
[certs] apiserver serving cert is signed for DNS names [ip-172-31-37-198 kubernetes kubernetes.default kubernet
es.default.svc kubernetes.default.svc.cluster.local] and IPs [10.96.0.1 172.31.37.198]
[certs] Generating "apiserver-kubelet-client" certificate and key
[certs] Generating "front-proxy-ca" certificate and key
[certs] Generating "front-proxy-client" certificate and key
[certs] Generating "etcd/ca" certificate and key
[certs] Generating "etcd/server" certificate and key
[certs] etcd/server serving cert is signed for DNS names [ip-172-31-37-198 localhost] and IPs [172.31.37.198 12
7.0.0.1 ::1]
[certs] Generating "etcd/peer" certificate and key
[certs] etcd/peer serving cert is signed for DNS names [ip-172-31-37-198 localhost] and IPs [172.31.37.198 127.
0.0.1 ::1]
[certs] Generating "etcd/healthcheck-client" certificate and key
[certs] Generating "apiserver-etcd-client" certificate and key
[certs] Generating "sa" key and public key
```



```
luster
[bootstrap-token] Creating the "cluster-info" ConfigMap in the "kube-public" namespace
[kubelet-finalize] Updating "/etc/kubernetes/kubelet.conf" to point to a rotatable kubelet client certificate a
nd key
[addons] Applied essential addon: CoreDNS
[addons] Applied essential addon: kube-proxy

Your Kubernetes control-plane has initialized successfully!

To start using your cluster, you need to run the following as a regular user:

  mkdir -p $HOME/.kube
  sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
  sudo chown $(id -u):$(id -g) $HOME/.kube/config

Alternatively, if you are the root user, you can run:

  export KUBECONFIG=/etc/kubernetes/admin.conf

You should now deploy a pod network to the cluster.
Run "kubectl apply -f [podnetwork].yaml" with one of the options listed at:
  https://kubernetes.io/docs/concepts/cluster-administration/addons/

Then you can join any number of worker nodes by running the following on each as root:

kubeadm join 172.31.37.198:6443 --token gci45u.g7qowjugiqw1ynrb \
        --discovery-token-ca-cert-hash sha256:a984e32c6c7c4815973519ebb45d64bf44a23f4e48dbf1195d8f0e695ea9409e
```

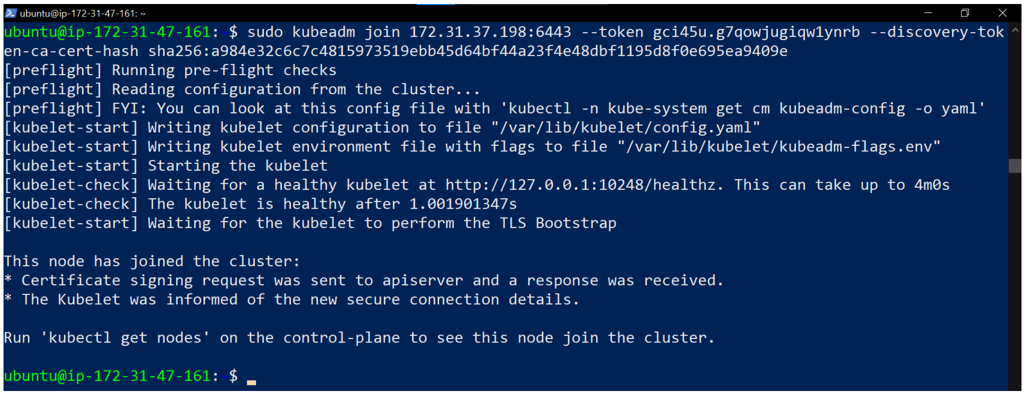
Run this command on master and also copy and save the Join command from above.
mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config



```
ubuntu@ip-172-31-37-198:~$ mkdir -p $HOME/.kube
ubuntu@ip-172-31-37-198:~$ sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
ubuntu@ip-172-31-37-198:~$ sudo chown $(id -u):$(id -g) $HOME/.kube/config
ubuntu@ip-172-31-37-198:~$ kubectl get nodes
NAME               STATUS     ROLES           AGE   VERSION
ip-172-31-37-198   NotReady   control-plane   12m   v1.31.1
```

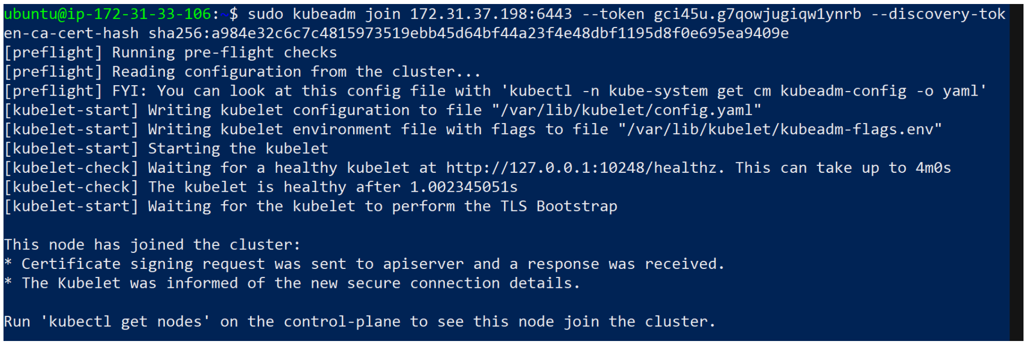
Connect master and worker nodes by running this command on the worker logged in terminals:
I ran the following command to do this,
kubeadm join 172.31.37.198:6443 --token gci45u.g7qowjugiqw1ynrb \ --discovery-token-ca-cert-hash sha256:a984e32c6c7c4815973519ebb45d64bf44a23f4e48dbf1195d8f0e695ea9409e

On node 1:



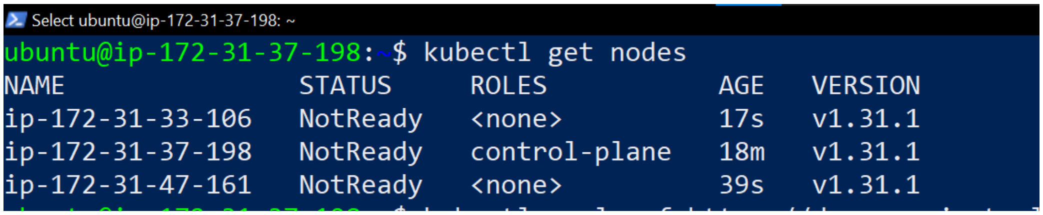
```
ubuntu@ip-172-31-47-161:~$ sudo kubeadm join 172.31.37.198:6443 --token gci45u.g7qowjugiqw1ynrb --discovery-tok
en-ca-cert-hash sha256:a984e32c6c7c4815973519ebb45d64bf44a23f4e48dbf1195d8f0e695ea9409e
[preflight] Running pre-flight checks
[preflight] Reading configuration from the cluster...
[preflight] FYI: You can look at this config file with 'kubectl -n kube-system get cm kubeadm-config -o yaml'
[kubelet-start] Writing kubelet configuration to file "/var/lib/kubelet/config.yaml"
[kubelet-start] Writing kubelet environment file with flags to file "/var/lib/kubelet/kubeadm-flags.env"
[kubelet-start] Starting the kubelet
[kubelet-check] Waiting for a healthy kubelet at http://127.0.0.1:10248/healthz. This can take up to 4m0s
[kubelet-check] The kubelet is healthy after 1.001901347s
[kubelet-start] Waiting for the kubelet to perform the TLS Bootstrap

This node has joined the cluster:
* Certificate signing request was sent to apiserver and a response was received.
* The Kubelet was informed of the new secure connection details.

Run 'kubectl get nodes' on the control-plane to see this node join the cluster.

ubuntu@ip-172-31-47-161:~$ 
```

On node 2:



```
ubuntu@ip-172-31-33-106:~$ sudo kubeadm join 172.31.37.198:6443 --token gci45u.g7qowjugiqw1ynrb --discovery-tok
en-ca-cert-hash sha256:a984e32c6c7c4815973519ebb45d64bf44a23f4e48dbf1195d8f0e695ea9409e
[preflight] Running pre-flight checks
[preflight] Reading configuration from the cluster...
[preflight] FYI: You can look at this config file with 'kubectl -n kube-system get cm kubeadm-config -o yaml'
[kubelet-start] Writing kubelet configuration to file "/var/lib/kubelet/config.yaml"
[kubelet-start] Writing kubelet environment file with flags to file "/var/lib/kubelet/kubeadm-flags.env"
[kubelet-start] Starting the kubelet
[kubelet-check] Waiting for a healthy kubelet at http://127.0.0.1:10248/healthz. This can take up to 4m0s
[kubelet-check] The kubelet is healthy after 1.002345051s
[kubelet-start] Waiting for the kubelet to perform the TLS Bootstrap

This node has joined the cluster:
* Certificate signing request was sent to apiserver and a response was received.
* The Kubelet was informed of the new secure connection details.

Run 'kubectl get nodes' on the control-plane to see this node join the cluster.
```

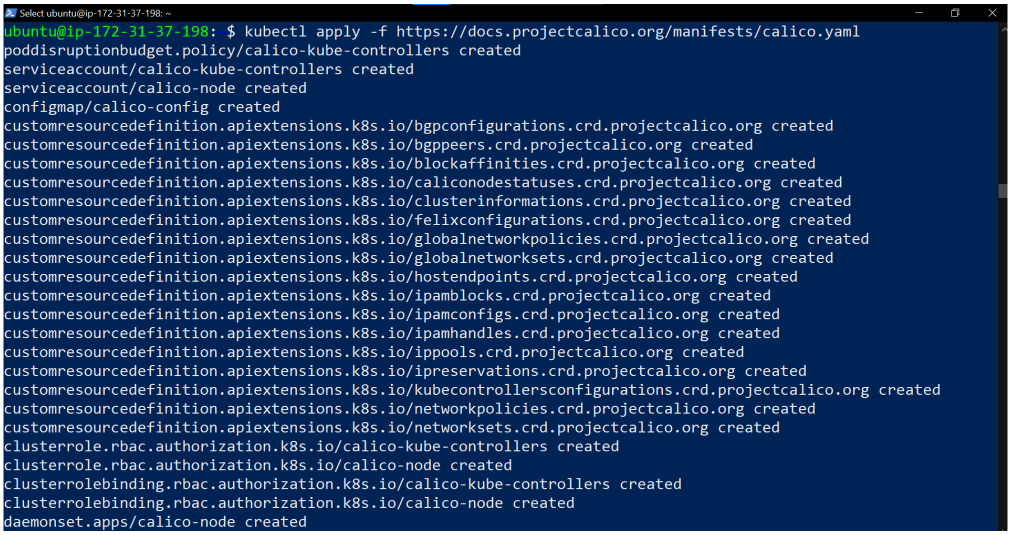
Run kubectl get nodes on the master instance to confirm the joining of the worker nodes



```
ubuntu@ip-172-31-37-198:~$ kubectl get nodes
NAME               STATUS     ROLES           AGE   VERSION
ip-172-31-33-106   NotReady   <none>          17s   v1.31.1
ip-172-31-37-198   NotReady   control-plane   18m   v1.31.1
ip-172-31-47-161   NotReady   <none>          39s   v1.31.1
```

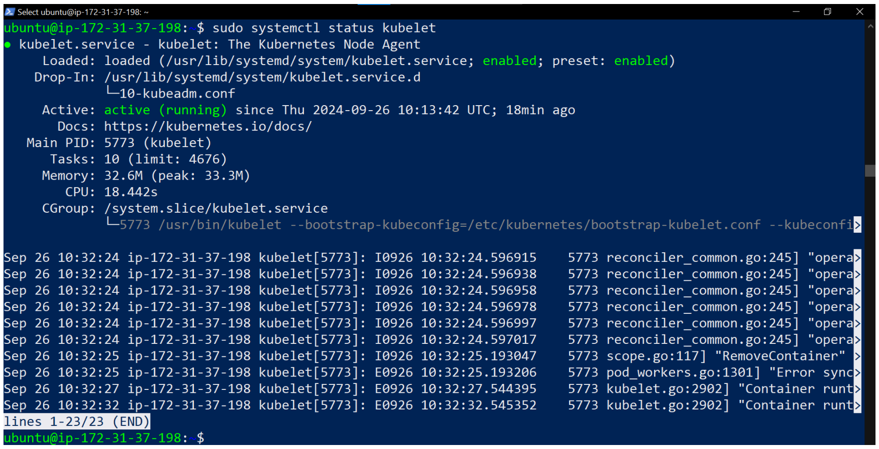
Since Status is NotReady we have to add a network plugin. And also we have to give the name to the nodes.
kubectl apply -f https://docs.projectcalico.org/manifests/calico.yaml

The command kubectl apply -f https://docs.projectcalico.org/manifests/calico.yaml deploys Calico, a networking and network security solution for Kubernetes, by applying the configuration specified in the provided YAML file from the Calico documentation. This sets up the necessary resources to enable networking capabilities within the Kubernetes cluster.



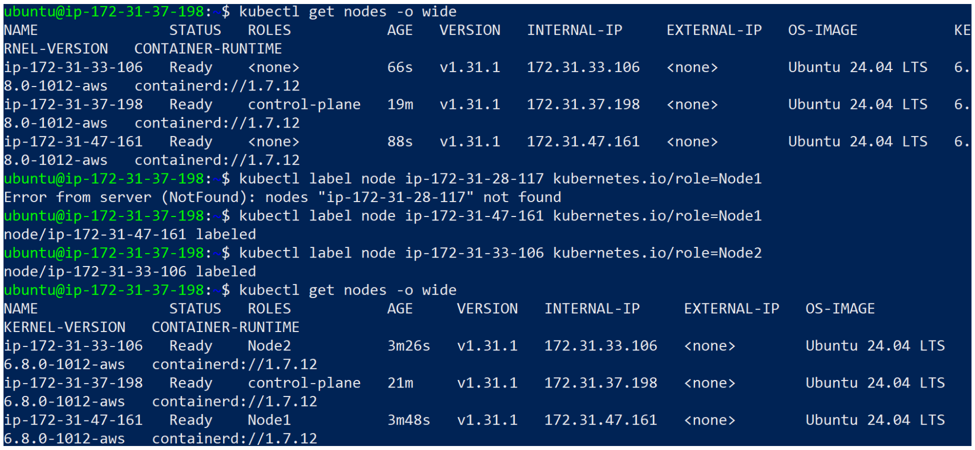
```
ubuntu@ip-172-31-37-198:~$ kubectl apply -f https://docs.projectcalico.org/manifests/calico.yaml
poddisruptionbudget.policy/calico-kube-controllers created
serviceaccount/calico-kube-controllers created
serviceaccount/calico-node created
configmap/calico-config created
customresourcedefinition.apiextensions.k8s.io/bgpconfigurations.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/bgppeers.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/blockaffinities.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/caliconodestatuses.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/clusterinformations.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/felixconfigurations.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/globalnetworkpolicies.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/globalnetworksets.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/hostendpoints.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/ipamblocks.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/ipamconfigs.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/ipamhandles.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/ippools.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/ipreservations.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/kubecontrollersconfigurations.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/networkpolicies.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/networksets.crd.projectcalico.org created
clusterrole.rbac.authorization.k8s.io/calico-kube-controllers created
clusterrole.rbac.authorization.k8s.io/calico-node created
clusterrolebinding.rbac.authorization.k8s.io/calico-kube-controllers created
clusterrolebinding.rbac.authorization.k8s.io/calico-node created
daemonset.apps/calico-node created
```

sudo systemctl status kubelet



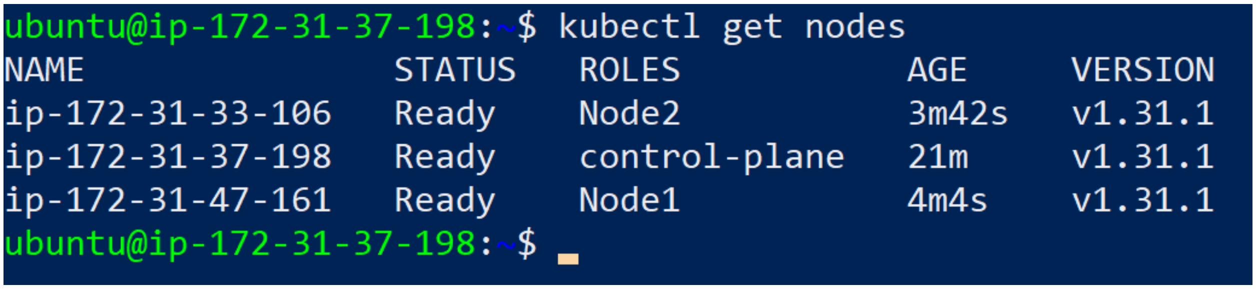
```
ubuntu@ip-172-31-37-198:~$ sudo systemctl status kubelet
● kubelet.service - kubelet: The Kubernetes Node Agent
     Loaded: loaded (/usr/lib/systemd/system/kubelet.service; enabled; preset: enabled)
    Drop-In: /usr/lib/systemd/system/kubelet.service.d
             └─10-kubeadm.conf
     Active: active (running) since Thu 2024-09-26 10:13:42 UTC; 18min ago
       Docs: https://kubernetes.io/docs/
   Main PID: 5773 (kubelet)
      Tasks: 10 (limit: 4676)
     Memory: 32.6M (peak: 33.3M)
        CPU: 18.442s
     CGroup: /system.slice/kubelet.service
             └─5773 /usr/bin/kubelet --bootstrap-kubeconfig=/etc/kubernetes/bootstrap-kubelet.conf --kubeconfi>

Sep 26 10:32:24 ip-172-31-37-198 kubelet[5773]: I0926 10:32:24.596915    5773 reconciler_common.go:245] "opera>
Sep 26 10:32:24 ip-172-31-37-198 kubelet[5773]: I0926 10:32:24.596938    5773 reconciler_common.go:245] "opera>
Sep 26 10:32:24 ip-172-31-37-198 kubelet[5773]: I0926 10:32:24.596958    5773 reconciler_common.go:245] "opera>
Sep 26 10:32:24 ip-172-31-37-198 kubelet[5773]: I0926 10:32:24.596978    5773 reconciler_common.go:245] "opera>
Sep 26 10:32:24 ip-172-31-37-198 kubelet[5773]: I0926 10:32:24.596997    5773 reconciler_common.go:245] "opera>
Sep 26 10:32:24 ip-172-31-37-198 kubelet[5773]: I0926 10:32:24.597017    5773 reconciler_common.go:245] "opera>
Sep 26 10:32:25 ip-172-31-37-198 kubelet[5773]: I0926 10:32:25.193047    5773 scope.go:117] "RemoveContainer" >
Sep 26 10:32:25 ip-172-31-37-198 kubelet[5773]: E0926 10:32:25.193206    5773 pod_workers.go:1301] "Error sync>
Sep 26 10:32:27 ip-172-31-37-198 kubelet[5773]: E0926 10:32:27.544395    5773 kubelet.go:2902] "Container runt>
Sep 26 10:32:32 ip-172-31-37-198 kubelet[5773]: E0926 10:32:32.545352    5773 kubelet.go:2902] "Container runt>
lines 1-23/23 (END)
ubuntu@ip-172-31-37-198:~$
```

kubectl get nodes -o wide helps us get to know that the Status is ready.



```
ubuntu@ip-172-31-37-198:~$ kubectl get nodes -o wide
NAME               STATUS   ROLES           AGE   VERSION   INTERNAL-IP     EXTERNAL-IP   OS-IMAGE           KE
RNEL-VERSION   CONTAINER-RUNTIME
ip-172-31-33-106   Ready    <none>          66s   v1.31.1   172.31.33.106   <none>        Ubuntu 24.04 LTS   6.
8.0-1012-aws   containerd://1.7.12
ip-172-31-37-198   Ready    control-plane   19m   v1.31.1   172.31.37.198   <none>        Ubuntu 24.04 LTS   6.
8.0-1012-aws   containerd://1.7.12
ip-172-31-47-161   Ready    <none>          88s   v1.31.1   172.31.47.161   <none>        Ubuntu 24.04 LTS   6.
8.0-1012-aws   containerd://1.7.12
ubuntu@ip-172-31-37-198:~$ kubectl label node ip-172-31-28-117 kubernetes.io/role=Node1
Error from server (NotFound): nodes "ip-172-31-28-117" not found
ubuntu@ip-172-31-37-198:~$ kubectl label node ip-172-31-47-161 kubernetes.io/role=Node1
node/ip-172-31-47-161 labeled
ubuntu@ip-172-31-37-198:~$ kubectl label node ip-172-31-33-106 kubernetes.io/role=Node2
node/ip-172-31-33-106 labeled
ubuntu@ip-172-31-37-198:~$ kubectl get nodes -o wide
NAME               STATUS   ROLES           AGE     VERSION   INTERNAL-IP     EXTERNAL-IP   OS-IMAGE
KERNEL-VERSION   CONTAINER-RUNTIME
ip-172-31-33-106   Ready    Node2           3m26s   v1.31.1   172.31.33.106   <none>        Ubuntu 24.04 LTS
6.8.0-1012-aws   containerd://1.7.12
ip-172-31-37-198   Ready    control-plane   21m     v1.31.1   172.31.37.198   <none>        Ubuntu 24.04 LTS
6.8.0-1012-aws   containerd://1.7.12
ip-172-31-47-161   Ready    Node1           3m48s   v1.31.1   172.31.47.161   <none>        Ubuntu 24.04 LTS
6.8.0-1012-aws   containerd://1.7.12
```

Or else run kubectl get nodes command



```
ubuntu@ip-172-31-37-198:~$ kubectl get nodes
NAME               STATUS   ROLES           AGE     VERSION
ip-172-31-33-106   Ready    Node2           3m42s   v1.31.1
ip-172-31-37-198   Ready    control-plane   21m     v1.31.1
ip-172-31-47-161   Ready    Node1           4m4s    v1.31.1
ubuntu@ip-172-31-37-198:~$ _
```

Conclusion: In this experiment, we successfully set up a Kubernetes cluster on AWS by creating the necessary infrastructure and configuring security groups. We installed Docker and Kubernetes components, initialized the master node, and connected the worker nodes. By deploying Calico for networking, we enabled effective communication within the cluster. This hands-on experience enhanced our understanding of Kubernetes architecture and equipped us with practical skills for managing containerized applications in a cloud environment.